Program: Mechatronics

Subject: Design of Mechatronics Systems(1) MCT 381

Handling Subsystem Report

Submitted to:

Dr. Shady Ahmed Maged

Assistant Prof. Omar M. Shehata

Submitted by: (Team 23)

Marwan Hesham 17P8173

Contents

List of Figures and Tables

Abstract

This report covers the handling subsystem by highlighting the expected behavior of the mentioned subsystem and the interaction of it with the feeding subsystem, assembly subsystem, storing and sorting subsystem and the disassembly subsystem respectively.

Introduction

The handling subsystem consists of two conveyors and plays one of the most important roles in the system since it interacts with all of the other subsystems. In our project the handling subsystem manages all the communication protocols that takes place, the particular reason for this circumstance is wire management.

System Requirements

Correct application of the VDI 2206 standard to insure High Production Rate with High accuracy and reliability.

Communication with each subsystem to offer a full integrated and automated design.

The feeding subsystem starts the assembly conveyor by placing the product on it. Both conveyors deliver the product to each subsystem and proceeds after that specific subsystem is done. The product should be centered on both conveyors whether by hardware or software means. Both conveyors are manipulated by respective subsystems to fit the subsystems need to get the task done.

Moreover, the disassembly conveyor starts the moment the storage and sorting subsystem's gripper places the product on the conveyor. Consequently, Disassembly conveyor can reverse direction upon the disassembly subsystem's request and the disassembled parts should return back to the feeding subsystem.

Project Plan

PROJECT TITLE	START DATE		
Production line	10/17	PROJECT DURATION	
	END DATE		Duration
Disassemble process	01/12		90 Days
TASK	**START DATE**	**END DATE**	**DURATION** in days
Project Topic announced	**10/17**	**10/20**	**4**
Brainstorming	**10/20**	**10/26**	**7**
Initial CAD design	**10/26**	**11/02**	**8**
Development of CAD design	**11/03**	**11/09**	**7**
Actuation Sizing of Systems	**11/10**	**11/16**	**7**
Presentation and Flow Chart	**11/17**	**11/23**	**7**
Manufacturing Processes	**11/24**	**11/30**	**7**
Electronic and Pneumatic Component Integration	**12/01**	**12/04**	**4**
Assembly of each Subsystem	**12/05**	**12/11**	**7**
Initial Integration	**12/12**	**12/20**	**9**
Integration Diagnosis	**12/21**	**12/28**	**8**
Initial Automation	**12/29**	**01/04**	**7**
Automation Diagnosis	**01/05**	**01/09**	**5**
Full Integration	**01/09**	**01/12**	**4**
Final Video	**01/12**	**01/12**	**1**
Report	**01/25**	**01/30**	**6**

Table 1 Project Plan

Mechanical Design

- ### Concept Design

This subsystem consists of an assembly conveyor to transport the parts from the feeding subsystem to the assembly subsystem and then to the storing and sorting subsystem and a disassembly conveyor to transport the assembled product from the storing and sorting subsystem to the disassembly subsystem and then back to the feeding subsystem.

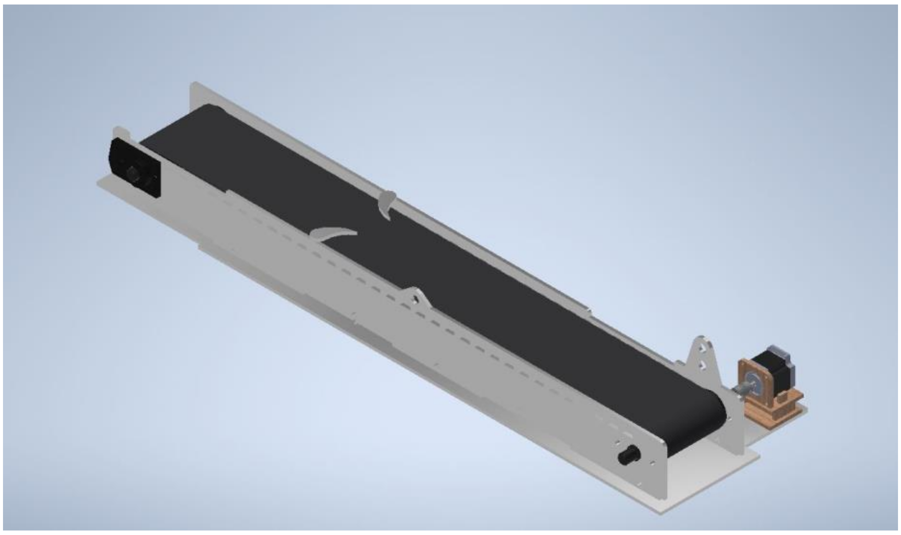


Figure 1 Assembly Conveyor

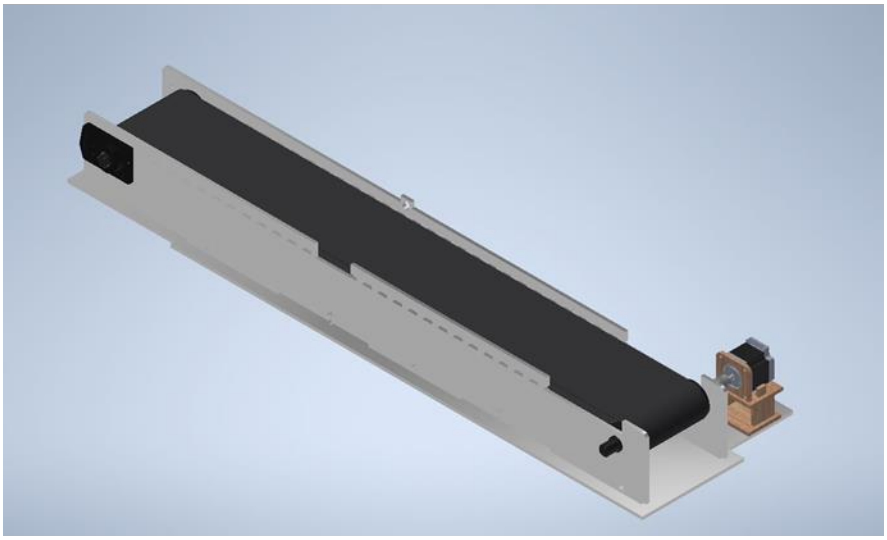


Figure 2: Disassembly Conveyor

- Conveyor Walls and Base (Assembly Conveyer)

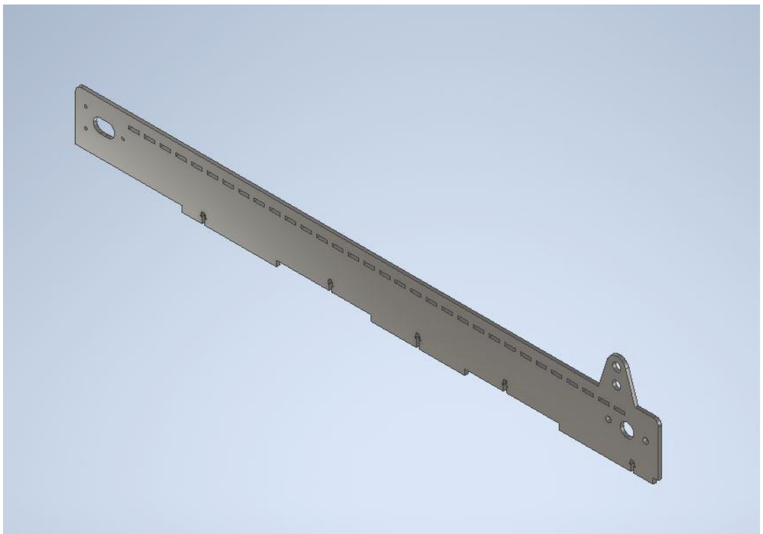


Figure 3 Assembly Subsystem Side

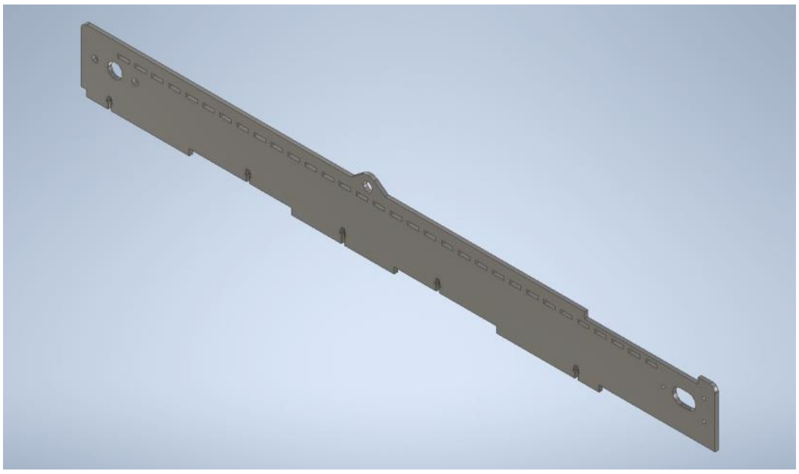


Figure 4: Feeding Subsystem Side

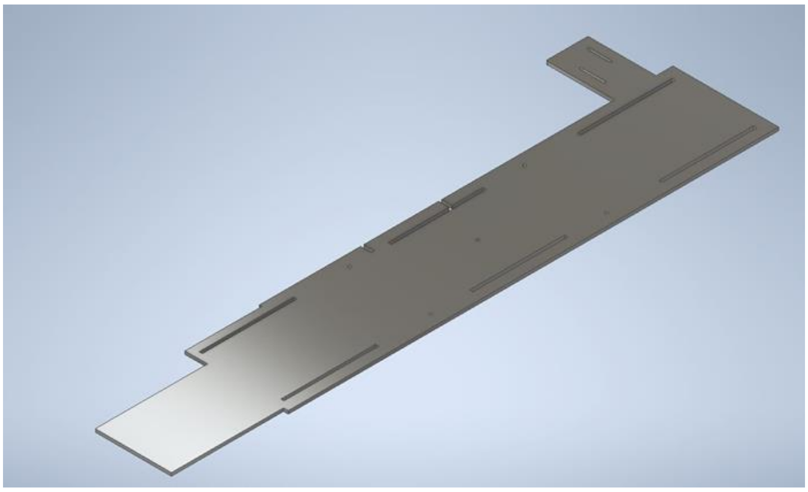


Figure 5 Assembly Conveyer Base

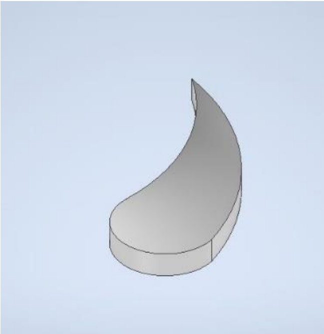


Figure 6 Centering Piece

- Conveyor Walls and Base (Disassembly Conveyer)

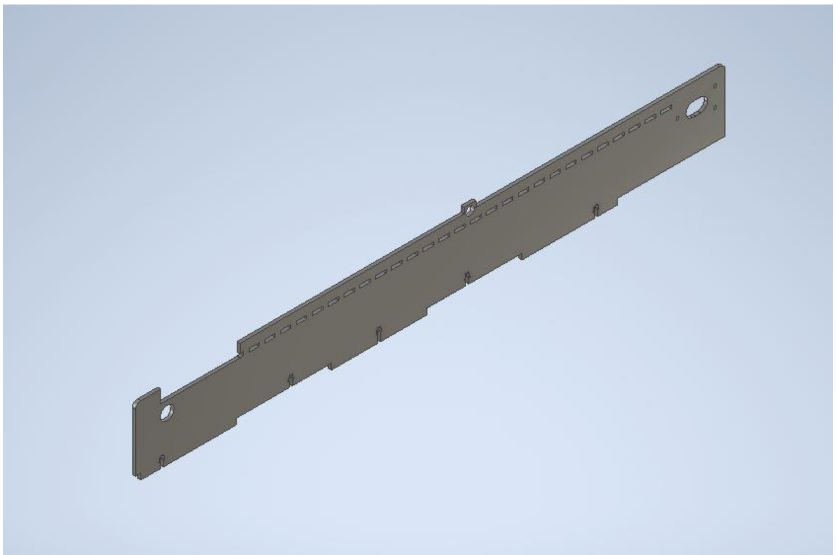


Figure 7 Disassembly Subsystem Side

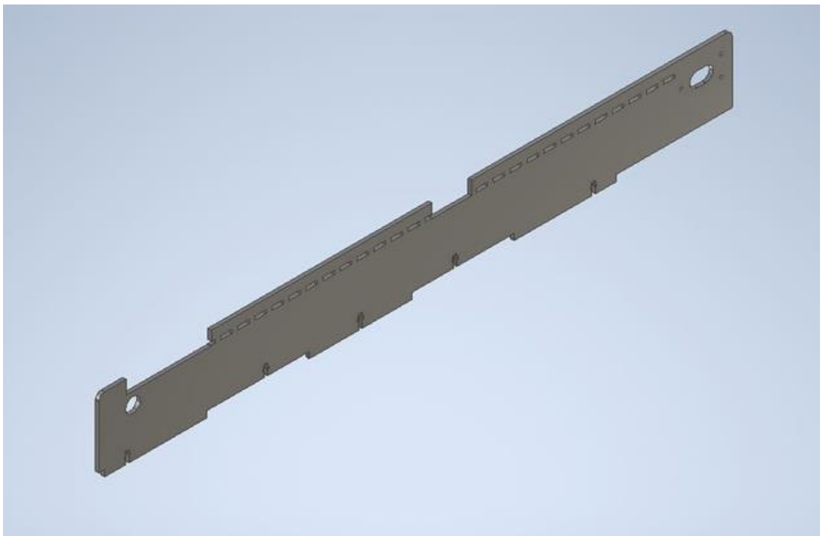


Figure 8 Return Feeding Subsystem Side

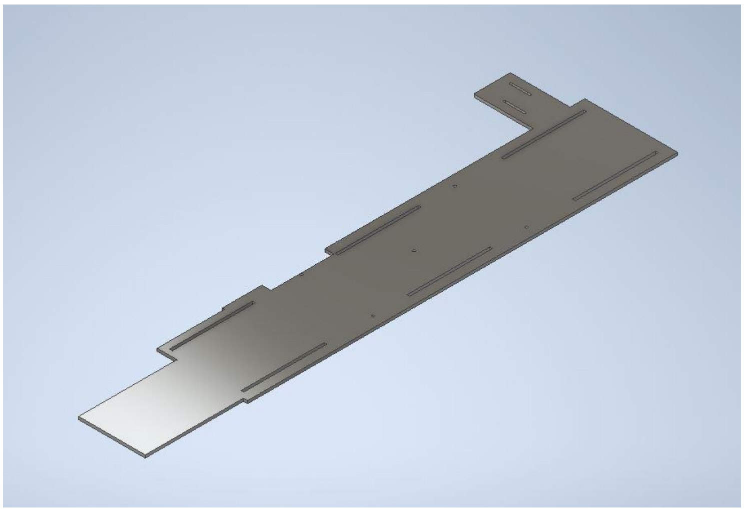


Figure 9 Disassembly Conveyor Base

- Conveyor Pulley

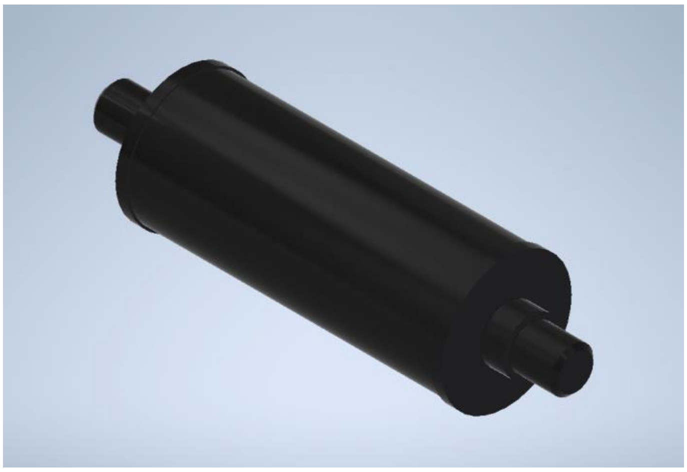


Figure 10: Free Pulley

Figure 11: Motor Pulley

- Conveyor Belt

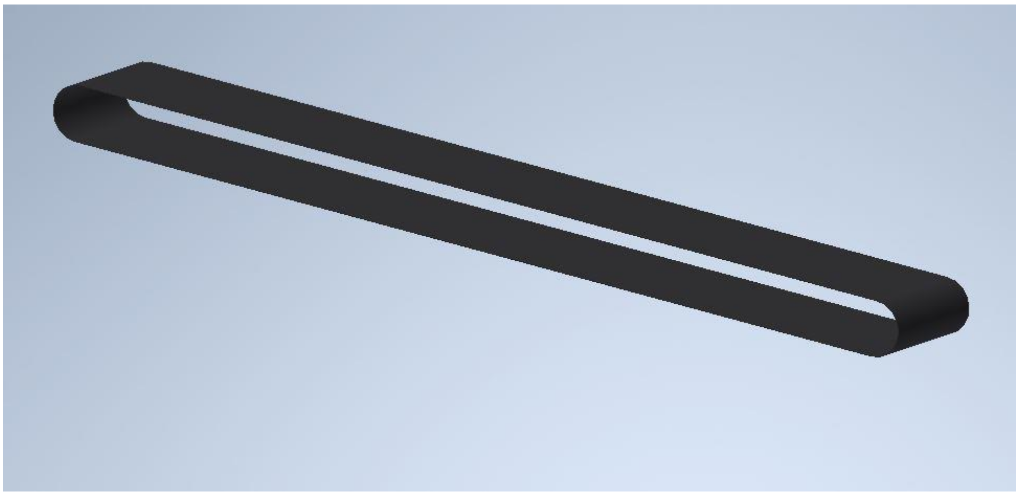


Figure 12: Conveyor Belt

- Support Panels (20mm x 6mm x 162mm)

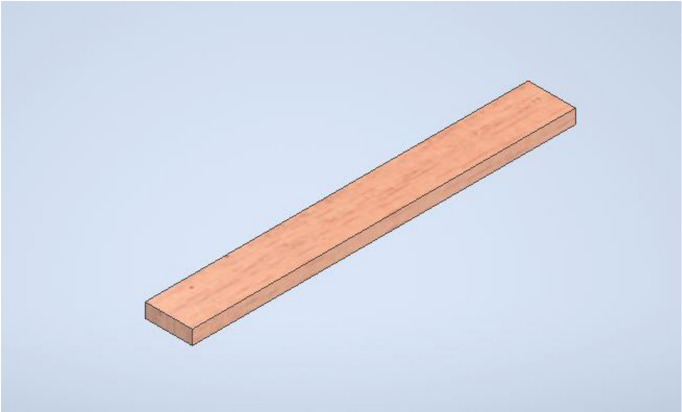


Figure 13 Support Panels (20mm x 6mm x 162mm)

- Belt Tensioner (3D Part)

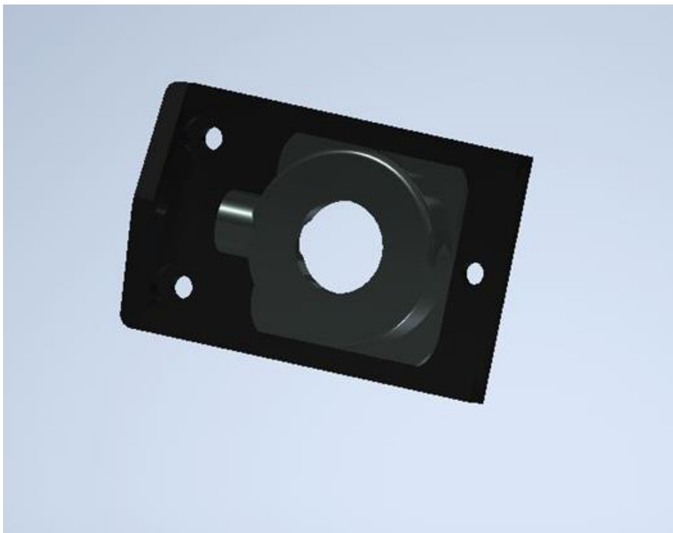


Figure 14 Belt Tensioner

- Flexible Coupling (10mm x 6.35mm)

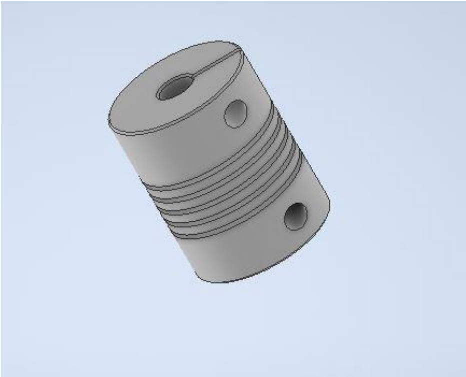


Figure 15 Flexible Coupling (10mm x 6.35mm)

- Nema-23 Motor and Motor Seat

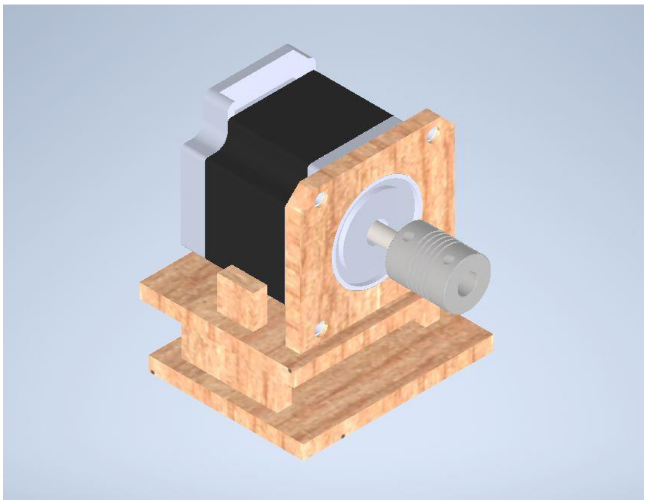


Figure 16 Assembly Conveyor Motor and Motor Seat

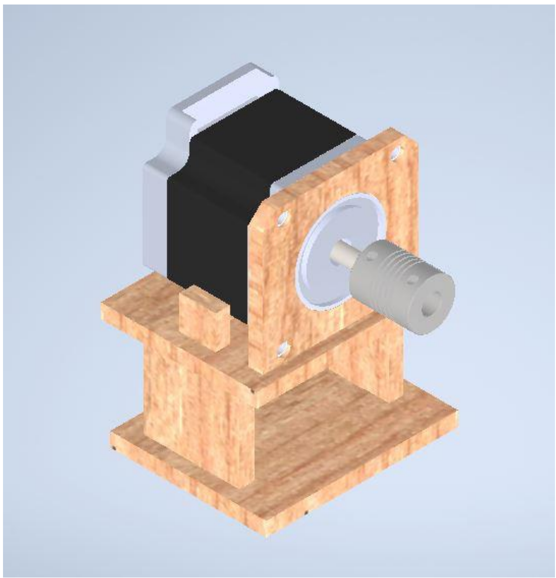


Figure 17 Disassembly Conveyor Motor and Motor Seat

- MATLAB Simulation

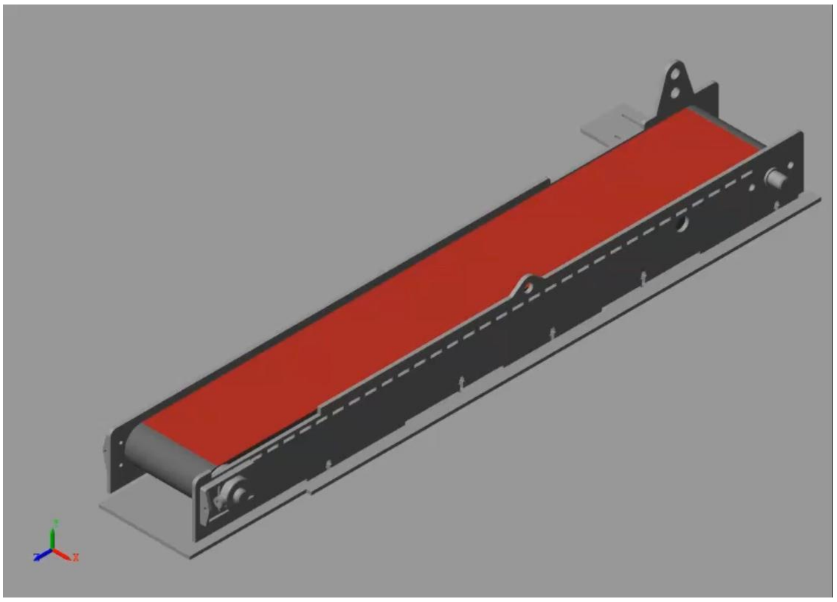


Figure 18 MATLAB Simulation

Manufacturing Processes

Initially, the conveyors were designed to have two levels of inclination so that the disassembly conveyor would be higher than the feeding subsystem so the pneumatic cylinder can push the product from the disassembly conveyor onto the feeding table slots. Similarly, the assembly conveyor would be lower so that the pneumatic cylinder can push the product onto the conveyor while retracting. However, this version was rejected, an easier solution was implemented which was to elevate the center of the disassembly conveyor pulley and lowering the center of the assembly conveyor pulley.

During the full CAD integration some dimensions were changed and with the other subsystems going through different versions, the conveyors had to be adjusted several times as they interact with all the other subsystems to ensure accurate position of sensors and interaction, before the conveyors are manufactured.

- Actuator Sizing

The actuator sizing calculation for both conveyors were done manually

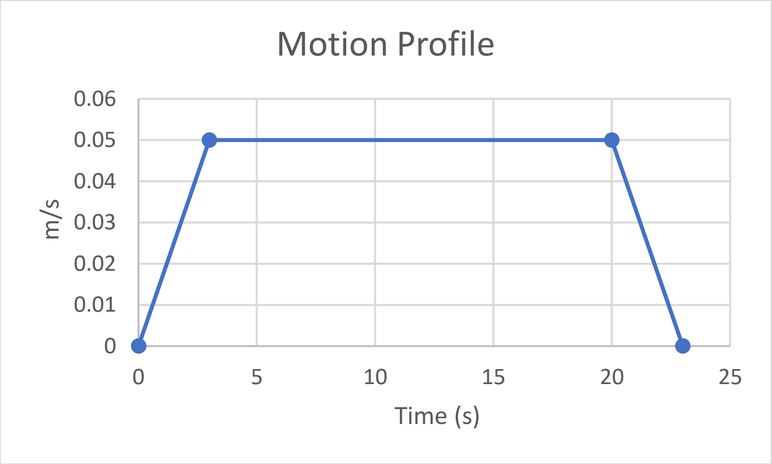
$$Assuming\ efficiency\ of\ belt\ \delta = 98\%$$

$$r1 = radius\ of\ pulley\ (mm)$$
$$T_c = torque\ required$$
$$m_p = mass\ of\ moved\ load + belt$$
$$\mu = 0.01$$
$$\alpha = \frac{2\pi N}{60t}$$

$$T_c = \frac{F_a(r_1)}{1000 * \delta}$$
$$F_a = m_p * g * \mu$$
$$T_a = T_c + T_{acc}$$
$$T_{acc} = J_t * \alpha$$
$$J_t = J_m + J_c + J_{p1} + J_{p2} + J_l$$
$$J_L = (m_L + m_b)(r_1)^2 * 10^{-6}$$

$$T_d = T_c - T_{acc}$$
$$T_{RMS} = \sqrt{T_a^2 * t_a + T_c^2 * t_c + T_d^2 * t_c}$$

$$J_{p1} = J_{p2} = 2.44 * 10^{-5} Kg\ m^2$$
$$J_L = (1.119)(30)^2(10^{-2}) = 1.0071 * 10^{-3}\ Kg\ m^2$$
$$J_t = 2\ (2.44 * 10^{-5}) + 1.0071 * 10^{-3} = 1.0559 * 10^{-3}\ Kgm^2$$
$$V = \omega\, r$$
$$\omega = \frac{2\pi N}{60}$$
$$N = 15.9\ rpm$$
$$\alpha = \frac{2\pi(15.9)}{60 * 3} = \frac{5}{9}$$



$$m_p = 2\,(\,0.083 + 0.022 + 0.022 + 0.048) + 0.769 = 1.119\ Kg$$

$$F_a = (1.119)(9.81)(0.01) = 1.1\ N$$

$$T_c = \frac{(1.1 * 30)}{1000 * 0.98} = 33.67 * 10^{-3} Nm$$

$$T_{acc} = J_t * \ \alpha = 1.0559 * 10^{-3} \left(\frac{5}{9}\right) = 5.866 * 10^{-4} Nm$$

$$T_a = \ T_c + \ T_{acc}$$

$$T_a = 34.2566 * 10^{-3} Nm$$

$$T_d = \ T_c - \ T_{acc}$$

$$T_d = 33.0834 * 10^{-3} Nm$$

$$T_{RMS} = \ \sqrt{(34.2566 * 10^{-3})^2(3) + (33.67 * 10^{-3})^2(17) + (33.0834 * 10^{-3})^2(3)}$$

$$= 161.66\ Nmm$$

The nema-17 stepper motor has a holding torque of 156.77 Nmm so we chose the next motor with higher holding torque which was nema 23 stepper motor.

Electrical Design

- ### Circuit Schematic Diagram

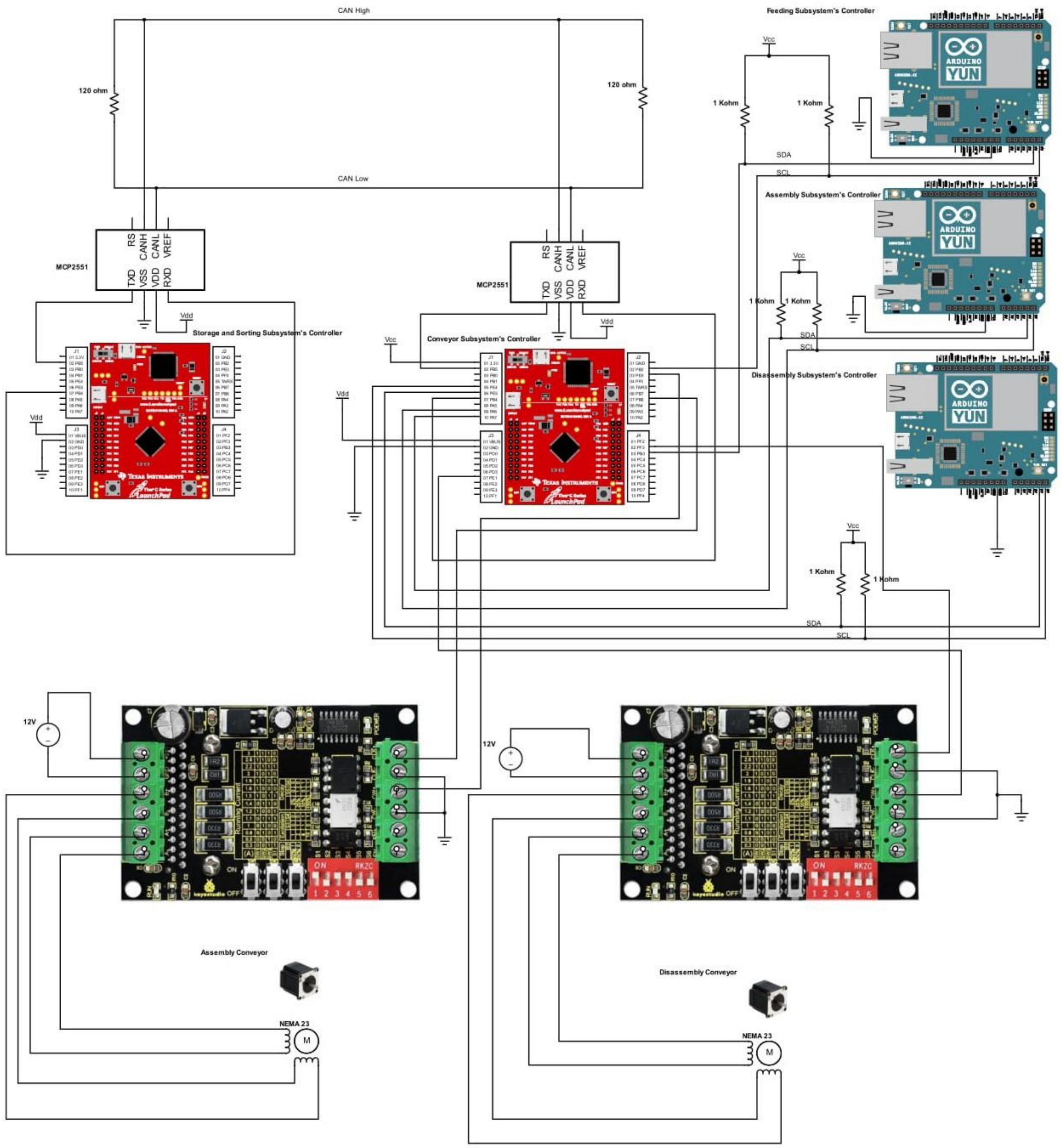


Figure 19: Circuit Schematic Diagram

The clearer version of the circuit is attached with the report here.

- Code
    - init.h

```c
#ifndef INIT_H
#define INIT_H

#include <math.h>
#include <stdint.h>
#include <string.h>
#include "tm4c123gh6pm.h"
#include "driverlib/pin_map.h"
#include <stdbool.h>
#include "inc/hw_gpio.h"
#include "inc/hw_types.h"
#include "inc/hw_memmap.h"
#include "inc/hw_i2c.h"
#include "inc/hw_can.h"
#include "driverlib/can.h"
#include "driverlib/sysctl.h"
#include "driverlib/pin_map.h"
#include "driverlib/gpio.h"
#include "driverlib/pwm.h"
#include "driverlib/i2c.h"
#include "driverlib/interrupt.h"
#include "driverlib/timer.h"
#include "driverlib/rom.h"
#include "driverlib/rom_map.h"
#include "driverlib/uart.h"
#include "inc/hw_ints.h"

#define GPIO_Direction  GPIO_PORTE_BASE
#define PINS GPIO_PIN_0 | GPIO_PIN_1
#define AssDIR  GPIO_PIN_0
#define DisDIR  GPIO_PIN_1
#define DIR   AssDIR | DisDIR
#define PWM_FREQUENCY 20

volatile uint32_t AssembConv;
volatile uint32_t DisAssemblyUnit;
volatile uint32_t DisAssembConv;

void StartAssemblyMotor() {
        GPIOPinWrite(GPIO_Direction,AssDIR,~AssDIR);
// ENABLES THE PWM_GEN_0 GENERATION
        ROM_PWMGenEnable(PWM0_BASE, PWM_GEN_0);
}
void StartDisassemblyMotor() {
        GPIOPinWrite(GPIO_Direction,DisDIR,~DisDIR);
// ENABLES THE PWM_GEN_3 GENERATION
        ROM_PWMGenEnable(PWM1_BASE, PWM_GEN_3);
}
void initClock(void){//Clock initialization
        SysCtlClockSet(SYSCTL_OSC_MAIN | SYSCTL_XTAL_16MHZ | SYSCTL_USE_PLL | SYSCTL_SYSDIV_5);
}
void initGPIO(void){//GPIO initialization and interrupt enabling
        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
        GPIOPinTypeGPIOOutput(GPIO_Direction, DIR);
}
```

```
void stopAssemblyStepper(void) {
        ROM_PWMGenDisable(PWM0_BASE, PWM_GEN_0);
        GPIOPinWrite(GPIO_PORTB_BASE,GPIO_PIN_0,0);
}
void stopDisassemblyStepper(void) {
        ROM_PWMGenDisable(PWM1_BASE, PWM_GEN_3);
        GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_3,0);
}
void InitAssemblyConveyer(){
            volatile uint32_t ui32Load;
    volatile uint32_t ui32PWMClock;
    volatile uint8_t ui8Adjust;
    ui8Adjust = 83;
    GPIOPinWrite(GPIO_Direction,AssDIR,~AssDIR);
// SET PWM CLOCK BY: CLOCK_CPU
    ROM_SysCtlPWMClockSet(SYSCTL_PWMDIV_1);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM0); // ENABLES PWM
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB); // ENABLES PB
// DEFINE PWM IN PIN PB6
    ROM_GPIOPinTypePWM(GPIO_PORTB_BASE, GPIO_PIN_6);
    ROM_GPIOPinConfigure(GPIO_PB6_M0PWM0); // CONFIGS PB6 AS M0PWM0
// PUT IN A VARIABLE THE PWM's CLOCK
    ui32PWMClock = SysCtlClockGet() / 64;
// TRANSFORMS THE CLOCK TO WORK IN A COUNTER THAT INITIALIZE AT 0
    ui32Load = (ui32PWMClock / PWM_FREQUENCY) - 1;
    // CONFIGS THE COUNTER AS DESCENT
    PWMGenConfigure(PWM0_BASE, PWM_GEN_0, PWM_GEN_MODE_DOWN);
    // SET THE COUNTER
    PWMGenPeriodSet(PWM0_BASE, PWM_GEN_0, ui32Load);
    // SPLIT THE COUNTER BY 1000 AND MULTIPLIES BY THE ADJUST
    ROM_PWMPulseWidthSet(PWM0_BASE, PWM_OUT_0, ui8Adjust * ui32Load / 1000);
    // CONFIGS THE PWM MODULE 0 AS OUT
    ROM_PWMOutputState(PWM0_BASE, PWM_OUT_0_BIT, true);
    // ENABLES THE PWM GENERATION
    ROM_PWMGenEnable(PWM0_BASE, PWM_GEN_0);
                stopAssemblyStepper();
                GPIOPinWrite(GPIO_Direction,AssDIR,~AssDIR);
}
void InitDisassemblyConveyer(){
                volatile uint32_t ui32Load;
    volatile uint32_t ui32PWMClock;
    volatile uint8_t ui8Adjust;
    ui8Adjust = 83;
    GPIOPinWrite(GPIO_Direction,DisDIR,~DisDIR);
// SET PWM CLOCK BY: CLOCK_CPU
    ROM_SysCtlPWMClockSet(SYSCTL_PWMDIV_1);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM1); // ENABLES PWM1
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF); // ENABLES PF
// DEFINE PWM IN PIN PF3
    ROM_GPIOPinTypePWM(GPIO_PORTF_BASE, GPIO_PIN_3);
    ROM_GPIOPinConfigure(GPIO_PF3_M1PWM7); // CONFIGS PF3 AS M1PWM7
    ui32PWMClock = SysCtlClockGet() / 64; // PUT IN A VARIABLE THE PWM's CLOCK
// TRANSFORMS THE CLOCK TO WORK IN A COUNTER THAT INITIALIZE AT 0
    ui32Load = (ui32PWMClock / PWM_FREQUENCY) - 1;
// CONFIGS THE COUNTER AS DESCENT
    PWMGenConfigure(PWM1_BASE, PWM_GEN_3, PWM_GEN_MODE_DOWN);
    // SET THE COUNTER
    PWMGenPeriodSet(PWM1_BASE, PWM_GEN_3, ui32Load);
    // SPLIT THE COUNTER BY 1000 AND MULTIPLIES BY THE ADJUST
```

```
    ROM_PWMPulseWidthSet(PWM1_BASE, PWM_OUT_7, ui8Adjust * ui32Load / 1000);
    // CONFIGS THE PWM MODULE 1 AS OUT
    ROM_PWMOutputState(PWM1_BASE, PWM_OUT_7_BIT, true);
    // ENABLES THE PWM GENERATION
    ROM_PWMGenEnable(PWM1_BASE, PWM_GEN_3);
                stopDisassemblyStepper();
}
void stepDisassemblyBackward(double mm) {
        int n;
        double rev;
        rev=mm/(360);
        rev=rev*1200;
        stopDisassemblyStepper();
        GPIOPinWrite(GPIO_Direction,DisDIR,DisDIR);
        for (n=0; n<rev; n++) {
                ROM_PWMGenEnable(PWM1_BASE, PWM_GEN_3);
                SysCtlDelay(10000);
        }
        stopDisassemblyStepper();
}
void motorDisassemblyBackward(){
        stopDisassemblyStepper();
        GPIOPinWrite(GPIO_Direction,DisDIR,DisDIR);
        ROM_PWMGenEnable(PWM1_BASE, PWM_GEN_3);
        SysCtlDelay(10000);
}
void motorAssemblyForward(){
        stopAssemblyStepper();
        GPIOPinWrite(GPIO_Direction,AssDIR,~AssDIR);
        ROM_PWMGenEnable(PWM0_BASE, PWM_GEN_0);
        SysCtlDelay(10000);
}
void setConveyors(void){
                if(AssembConv==0)
                                stopAssemblyStepper();
                else if(AssembConv==1)
                                StartAssemblyMotor();
                if(DisAssembConv==1)
                                StartDisassemblyMotor();
                else if(DisAssembConv==0)
                                stopDisassemblyStepper();
}
#endif
```

- I2cConnection.h

```
#ifndef I2CCONNECTION_H
#define I2CCONNECTION_H

#include "init.h"
#define SLAVE_ADDRESS          4

static uint32_t g_ui32DataRx=0XFF;

//Disassembly Subsystem Subroutine
void I2C2SlaveIntHandler(void){
    // Clear the I2C2 interrupt flag.
    I2CSlaveIntClear(I2C2_BASE);
                if(I2CSlaveStatus(I2C2_BASE)==I2C_SLAVE_ACT_TREQ){
            //Sends to the Disassembly Subsystem to know which
            //product arrived the tall(4) or the short(8)
                  I2CSlaveDataPut(I2C2_BASE,DisAssemblyUnit);
                }
                else{
                g_ui32DataRx =I2CSlaveDataGet(I2C2_BASE);
                if(g_ui32DataRx==3)
                        StartDisassemblyMotor();
                else if(g_ui32DataRx==2)
                        stopDisassemblyStepper();
                else if(g_ui32DataRx==5){
                        stepDisassemblyBackward(2000);
                        g_ui32DataRx=10;
                        }
                I2CSlaveStatus(I2C2_BASE);
        }
}
//Assembly Subsystem Subroutine
void I2C1SlaveIntHandler(void){
    // Clear the I2C1 interrupt flag.
    I2CSlaveIntClear(I2C1_BASE);
                g_ui32DataRx =I2CSlaveDataGet(I2C1_BASE);
                if(g_ui32DataRx==0)
                        stopAssemblyStepper();
                else if(g_ui32DataRx==1)
                        StartAssemblyMotor();
                I2CSlaveStatus(I2C1_BASE);
}
//Feeding Subsystem Subroutine
void I2C0SlaveIntHandler(void){
    // Clear the I2C0 interrupt flag.
    I2CSlaveIntClear(I2C0_BASE);
                g_ui32DataRx =I2CSlaveDataGet(I2C0_BASE);
                if(g_ui32DataRx==0)
                        stopAssemblyStepper();
                else if(g_ui32DataRx==1)
                        StartAssemblyMotor();
                Else if(g_ui32DataRx==3)
                        StartDisassemblyMotor();
                else if(g_ui32DataRx==2)
                        stopDisassemblyStepper();
                I2CSlaveStatus(I2C0_BASE);
}
void InitI2C0(void){
        ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C0);
        ROM_SysCtlPeripheralReset(SYSCTL_PERIPH_I2C0);
        ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
```

```
        ROM_GPIOPadConfigSet(GPIO_PORTB_BASE,GPIO_PIN_3|GPIO_PIN_2, GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD);
        ROM_GPIOPinConfigure(GPIO_PB2_I2C0SCL);
        ROM_GPIOPinConfigure(GPIO_PB3_I2C0SDA);
        ROM_GPIOPinTypeI2CSCL(GPIO_PORTB_BASE, GPIO_PIN_2);
        ROM_GPIOPinTypeI2C(GPIO_PORTB_BASE, GPIO_PIN_3);
        ROM_I2CMasterInitExpClk(I2C0_BASE,     MAP_SysCtlClockGet(), false);
        HWREG(I2C0_BASE + I2C_O_FIFOCTL) = 80008000;
        IntEnable(INT_I2C0);
        I2CSlaveIntEnableEx(I2C0_BASE, I2C_SLAVE_INT_DATA);
        I2CSlaveEnable(I2C0_BASE);
        I2CSlaveInit(I2C0_BASE, SLAVE_ADDRESS);
        I2CIntRegister(I2C0_BASE,I2C1SlaveIntHandler);
        IntPrioritySet(INT_I2C0,0);
}
void InitI2C1(void){
        ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C1);
        ROM_SysCtlPeripheralReset(SYSCTL_PERIPH_I2C1);
        //enable GPIO peripheral that contains I2C1
        ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
// Configure the pin muxing for I2C1 functions on port A6 and A7.
        ROM_GPIOPinConfigure(GPIO_PA6_I2C1SCL);
        ROM_GPIOPinConfigure(GPIO_PA7_I2C1SDA);
        // Select the I2C function for these pins.
        ROM_GPIOPinTypeI2CSCL(GPIO_PORTA_BASE, GPIO_PIN_6);
        ROM_GPIOPinTypeI2C(GPIO_PORTA_BASE, GPIO_PIN_7);
        //clear I2C FIFOs
        HWREG(I2C1_BASE + I2C_O_FIFOCTL) = 80008000;
        IntEnable(INT_I2C1);
        I2CSlaveIntEnableEx(I2C1_BASE, I2C_SLAVE_INT_DATA);
        I2CSlaveEnable(I2C1_BASE);
        I2CSlaveInit(I2C1_BASE, SLAVE_ADDRESS);
        I2CIntRegister(I2C1_BASE,I2C1SlaveIntHandler);
        IntPrioritySet(INT_I2C1,0);
}
void InitI2C2(void){
        ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C2);
        ROM_SysCtlPeripheralReset(SYSCTL_PERIPH_I2C2);
        //enable GPIO peripheral that contains I2C 2
        ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
// Configure the pin muxing for I2C2 functions on port E4 and E5.
        ROM_GPIOPinConfigure(GPIO_PE4_I2C2SCL);
        ROM_GPIOPinConfigure(GPIO_PE5_I2C2SDA);
        // Select the I2C function for these pins.
        ROM_GPIOPinTypeI2CSCL(GPIO_PORTE_BASE, GPIO_PIN_4);
        ROM_GPIOPinTypeI2C(GPIO_PORTE_BASE, GPIO_PIN_5);
        //clear I2C FIFOs
        HWREG(I2C2_BASE + I2C_O_FIFOCTL) = 80008000;
        IntEnable(INT_I2C2);
        I2CSlaveIntEnableEx(I2C2_BASE, I2C_SLAVE_INT_DATA);
        I2CSlaveEnable(I2C2_BASE);
        I2CSlaveInit(I2C2_BASE, SLAVE_ADDRESS);
        I2CIntRegister(I2C2_BASE,I2C2SlaveIntHandler);
        IntPrioritySet(INT_I2C2,0);
}
#endif
```

- CanConnection.h

```
#ifndef CANCONNECTION_H
#define CANCONNECTION_H

#include "init.h"

// msg recieved flag
volatile bool rxFlag = 0;
// error flag
volatile bool errFlag = 0;
// the CAN msg object
tCANMsgObject msg;
// 8-byte buffer for rx message data
unsigned char msgData[8];

// CAN interrupt handler
void CANIntHandler(void) {
        // read interrupt status
unsigned long status = CANIntStatus(CAN0_BASE, CAN_INT_STS_CAUSE);
        // controller status interrupt
        if(status == CAN_INT_INTID_STATUS) {
                status = CANStatusGet(CAN0_BASE, CAN_STS_CONTROL);
                errFlag = 1;
        }
        // msg object 1
        else if(status == 1) {
                // clear interrupt
                CANIntClear(CAN0_BASE, 1);
                // set rx flag
                rxFlag = 1;
                // clear any error flags
                errFlag = 0;
        }
}
void InitCAN0(void){
        // Set up CAN0
        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
        GPIOPinConfigure(GPIO_PB4_CAN0RX);
        GPIOPinConfigure(GPIO_PB5_CAN0TX);
        GPIOPinTypeCAN(GPIO_PORTB_BASE, GPIO_PIN_4 | GPIO_PIN_5);
        SysCtlPeripheralEnable(SYSCTL_PERIPH_CAN0);
        CANInit(CAN0_BASE);
        CANBitRateSet(CAN0_BASE, SysCtlClockGet(), 500000);
        CANIntRegister(CAN0_BASE, CANIntHandler);
        CANIntEnable(CAN0_BASE,   CAN_INT_MASTER   |   CAN_INT_ERROR
CAN_INT_STATUS);
        IntEnable(INT_CAN0);
        CANEnable(CAN0_BASE);
        msg.ui32MsgID = 0;
        msg.ui32MsgIDMask = 0;
        msg.ui32Flags=MSG_OBJ_RX_INT_ENABLE | MSG_OBJ_USE_ID_FILTER;
        msg.ui32MsgLen = 8; // allow up to 8 bytes
        // Load msg into CAN peripheral message object 1
        //so, it can trigger interrupts on any matched rx messages
        CANMessageSet(CAN0_BASE, 1, &msg, MSG_OBJ_TYPE_RX);
}
#endif
```

- Main.c

```c
#include "I2cConnection.h"
#include "CanConnection.h"

//Main routine
int main(void) {
  initClock();
  initGPIO();
  InitI2C1();
  InitI2C2();
  IntMasterEnable();
  InitAssemblyConveyer();
  InitDisassemblyConveyer();
  StartDisassemblyMotor();
  StartAssemblyMotor();
  while(1){
  // rx interrupt has occured
          if(rxFlag) {
          // set pointer to rx buffer
                  msg.pui8MsgData = msgData;
                   // read CAN message object 1 from CAN peripheral
                  CANMessageGet(CAN0_BASE, 1, &msg, 0);
                  // clear rx flag
                  rxFlag = 0;
                  AssembConv = msgData[0];
                  DisAssemblyUnit= msgData[1];
                  DisAssembConv= msgData[2] ;
          }
                  setConveyors();
      }
  }
```

- Communication Protocol

The assembly and disassembly conveyors are controlled by one microcontroller (TM4C123GH6PM) which is connected to the assembly subsystem’s microcontroller (Arduino Uno) and the disassembly subsystem’s microcontroller (Arduino Uno) each by an Inter-integrated circuit (I2C); however, the conveyors microcontroller is connected to the storage and sorting subsystem by means of Controller Area Network (CAN).

Software Flowchart

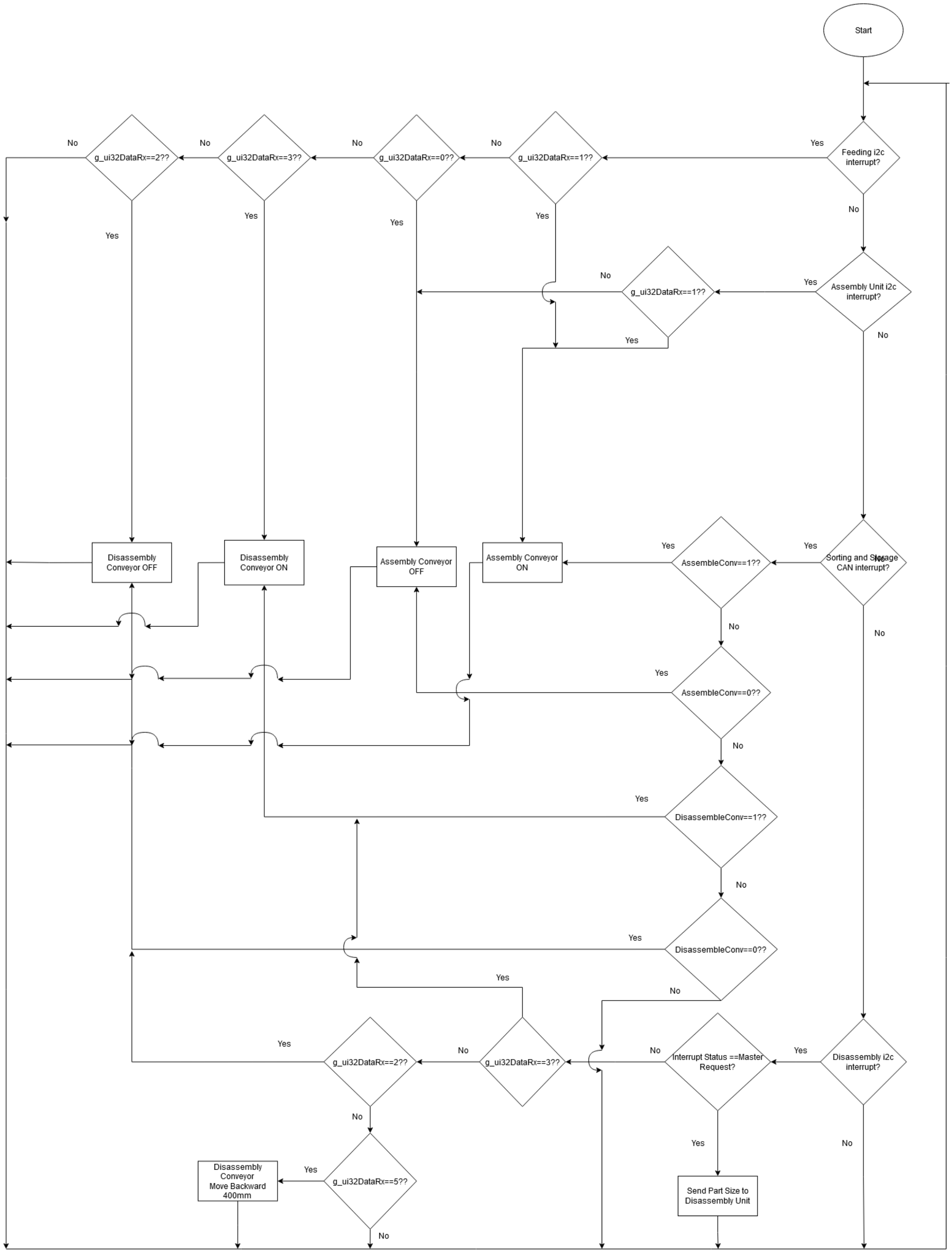


Figure 20 Software Flowchart

The clearer version of the flowchart is attached with the report here.

Implementation Efforts

Firstly, the standard parts such as bearings, fasteners and electronics were bought and accordingly, the artelon was bought and machined on the turning machine through turning processes and roughing to produce the pulleys.

Furthermore, the belt was bought, and the belt tensioners were manufactured through 3d printing.

While, the conveyors' walls and base and centered parts were all laser cut parts from a 6mm wood sheet that went through a spraying phase to achieve the final look. Consequently, all the parts were assembled, and the conveyors were configured by connecting the stepper motor to the pulley by a flexible coupling and adjusting the tensioner until slipping is minimized.

Both conveyors were assembled and tested separately then integrated with the respective subsystems to adjust position, adding the sensors, and adjusting the IR proximity sensors range and pneumatic cylinders range.

Manufactured Design

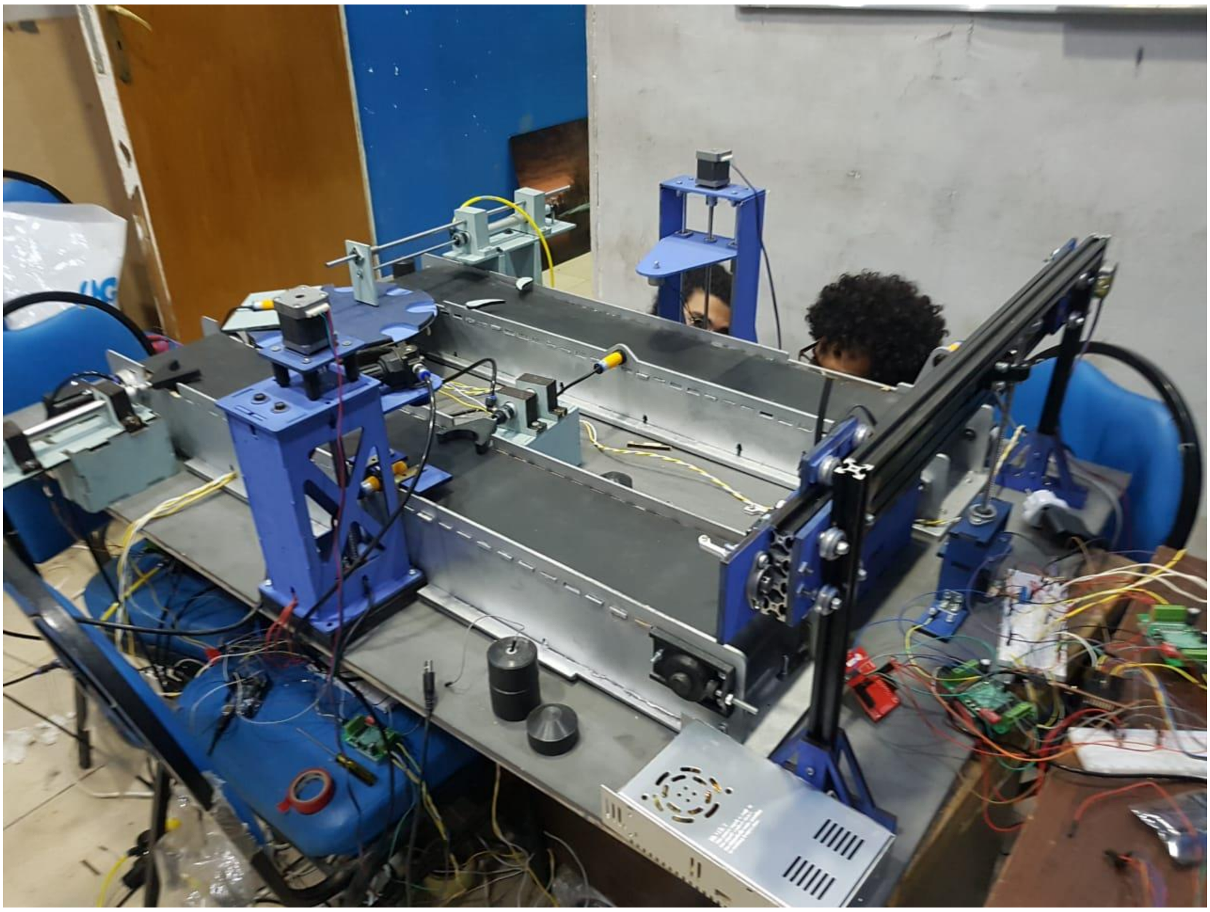


Figure 21 Manufactured Conveyor in full assembly

Bills of Materials

BILL OF MATERIALS

Item to be created:
Handling Subsystem

COMPONENT	BASE QTY	COST PER UNIT (L.E.)	SUBTOTAL (L.E.)
Nema23 Stepper Motor	2	65.00	130.00
MCP2551(CAN Chip)	2	25.00	50.00
Belts	2	180	360.00
tb6560 Driver	2	185.00	370.00
Self-Aligning Flange Bearing (10mm)	4	45.00	180.00
Wiring (Jumpers)	-	60.00	60.00
Flexible Coupling (10mm x 6.35mm)	2	45	90.00
Artelon	-	200.00	200.00
Fasteners	40	0.75	30.00
Wood (6mm)	-	250	250.00
Tiva C (TM4C123GH6PM)	1	600.00	600.00
3D Printing	-	210.00	210.00
Laser Cut parts	-	300.00	300.00
Turning	-	350	350.00
Spray Paint	8	15.00	120.00

Table 2 BOM

TOTAL COST (L.E.):	3300.00